CS60003 Algorithm Design and Analysis, Autumn 2010–11

End-Semester Examination

Maximum marks: 100 November 21, 2010 (FN) Total time: 3 hours

---

Roll no: ____________ **Name:** ______________________________

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. The knapsack problem discussed in the class is an optimization problem. Consider the following decision version of the knapsack problem. Given n objects $O_1, O_2, \ldots, O_n$ with respective weights $w_1, w_2, \ldots, w_n$ and with respective profits $p_1, p_2, \ldots, p_n$, and given a knapsack of capacity C and a profit bound P, decide whether there exists a subcollection $O_{i_1}, O_{i_2}, \ldots, O_{i_k}$ of the given objects such that $\sum_{j=1}^{k} w_{i_j} \leqslant C$ (knapsack capacity cannot be exceeded) and $\sum_{j=1}^{k} p_{i_j} \geqslant P$ (at least a profit of P can be made).

(a) Prove that the decision version of the knapsack problem can be solved in polynomial time if and only if the optimization version of the knapsack problem can be solved in polynomial time. **(10)**

(b) Prove that the decision version of the knapsack problem is NP-Complete.

(Hint: You may use the partition problem which, given positive integers $a_1, a_2, \ldots, a_n$ with $A = \sum_{i=1}^{n} a_i$, decides whether there exists a subcollection $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ with $\sum_{j=1}^{k} a_{i_j} = A/2$.) **(10)**

2. A cut in an undirected graph $G = (V, E)$ is a partition of V in two (disjoint) subsets S, T. Define by $E(S, T)$ the set of all edges of G with one endpoint in S and the other in T. The MAX-CUT problem is an optimization problem that determines a cut S, T for which the size of the set $E(S, T)$ (the number of cross edges) is as large as possible.

Recall that in the Ford-Fulkerson algorithm, we have dealt with *minimum* cuts in order to solve the dual problem of maximizing network flow. The Ford-Fulkerson algorithm is not truly polynomial-time, but has variants that run in polynomial time in the input size. The MAX-CUT problem (more correctly, a suitable decision version of this problem), on the other hand, is NP-Complete (you are not asked to prove this).

Prof. Myopia proposes the following approximation algorithm for solving the MAX-CUT problem.

```
1. Start with an arbitrary partition $S, T$ of $V$.
2. Repeat the following two steps until no further vertex movement is possible:
    (a) For each vertex $v \in S$, check whether the cut $(S-v, T+v)$ has more cross edges than $(S,T)$;
        and if so, delete $v$ from $S$ and include $v$ in $T$.
    (b) For each vertex $v \in T$, check whether the cut $(S+v, T-v)$ has more cross edges than $(S,T)$;
        and if so, delete $v$ from $T$ and include $v$ in $S$.
3. Return $S, T$.
```

(a) Prove that Prof. Myopia's algorithm runs in polynomial time (in the input size). **(5)**

(b) Prove or disprove: Prof. Myopia's algorithm outputs the optimal solution for bipartite graphs. **(5)**

(c) Prove that the approximation ratio of Prof. Myopia's algorithm is $1/2$. **(5)**

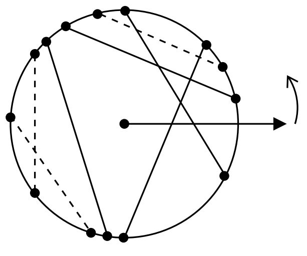
(d) Demonstrate that this approximation ratio is tight (suggest an *infinite* family of graphs). **(5)**

3. The subset-sum problem (SSP) decides whether a given collection of positive integers $a_1, a_2, \ldots, a_n$ has a subcollection whose elements add up to a given positive integer t. We proved that SSP is an NP-Complete problem. Recall also that an algorithm is called *pseudo-polynomial-time*, if its running time is a polynomial in the size of the *unary* representation of the input. We call an NP-Complete problem *weakly NP-Complete* if it admits a pseudo-polynomial-time algorithm. Prove that SSP is weakly NP-Complete.

(Hint: The knapsack problem might help you. Also note that the unary size of $a_1, a_2, \ldots, a_n$ is $n + \sum_{i=1}^{n} a_i$.) **(20)**

4. Consider the problem of finding the i-th smallest element in an array A of n integers. Ms. Lucky proposes the following randomized algorithm to solve this problem. She chooses a (uniformly) random element x of A. She then uses the partitioning algorithm of Quick Sort on A with respect to the pivot x. Suppose that x is placed in the k-th position after the partitioning (counting starts from 1). If $k = i$, the algorithm returns x. If $k > i$, then a recursive call is made on the smaller subarray (of size $k - 1$) and with the same i. Finally, if $k < i$, then a recursive call is made on the larger subarray (of size $n - k$) with i replaced by $i - k$. Deduce that the expected running time of Ms. Lucky's algorithm is $\mathrm{O}(n \log n)$. (Notice that this running time may depend upon i (in addition to n). In your calculations, you may suitably ignore this dependence.) **(20)**

5. You are given a set of n chords in a circle. Each chord may be viewed as an opaque piece of string. Your task is to determine which chords are visible (fully or partially) from the center. An example is given below. The dotted chords are the only chords that are not visible (not even partially) from the center.



(a) Propose an $O((n + h) \log n)$-time ray-sweep algorithm for solving this problem, where h is the total number of intersections of the given chords. Clearly describe the events in your algorithm and how they are handled. Assume that the chords are in general position, that is, no two of them share an endpoint, and no three of them are concurrent. **(10)**

(b) Mention relevant data structures that your algorithm uses (like the organization of the event queue and the sweep-ray information). **(5)**

(c) Deduce that the running time of your algorithm is $\mathrm{O}((n+h)\log n)$. **(5)**

For rough work. If used for continuation of answers from Pages 1–10, supply appropriate pointers earlier.